**Ausbildung Fachinformatiker Anwendungsentwicklung**

# Programmierübung (mit Musterlösung)

C–Programmierung

## 1 Aufgabe

Es ist ein Programm (`myfilter.c`) zu schreiben, das den Namen einer beliebigen Text–Datei/Pfad

1. als Parameter einliest,
2. diese Datei öffnet,
3. deren Inhalt einliest,
4. bestimmte Operationen ausführt und
5. das Ergebnis in die Standard–Ausgabe schreibt.

Der Datei/Pfadname kann entweder ohne Parameterkennzeichnung übergeben werden oder mit einer Parameterkennzeichnung (`myfilter -i <inputfile>` oder `myfilter <inputfile>`). Weitere optionale Parameter müssen von dem Programm eingelesen werden können. Die Parameter bestehen jeweils aus einem Buchstaben und sind durch ein vorangestelltes - gekennzeichnet. Diese Parameter bestimmen, welche von den u.g. Operationen ausgeführt werden sollen. Werden keine optionalen Parameter eingegeben, so soll eine `usage()`–Meldung ausgegeben werden (`./myfilter [-c|-w|-l] [-i] inputfile`). Folgende Operationen sollen durch das Programm durchgeführt werden können:

| Parameter | Operation | Ausgabe |
|---|---|---|
| kein | mache garnichts | `usage()` ausgeben |
| -c | Zähle alle Zeichen | Zeichen=nnnn |
| -w | Zähle alle Wörter | Wörter=nnnn |
| -l | Zähle alle Zeilen | Zeilen=nnnn |

Das Programm soll so konzipiert werden, dass später gegebenenfalls weitere optionale Parameter leicht hinzuzufügen sind. Außerdem soll das Programm als *Filter* arbeiten können, d.h. falls kein Datei oder Pfadname als Parameter angegeben wurde, soll das Programm seine Eingabe aus dem Kanal Standard–Eingabe beziehen. Das Programm soll seine Ausgabe immer in die Standard–Ausgabe schreiben. Einige mögliche Aufrufe könnten z.B. folgendermaßen aussehen:

```
bash$> ./myfilter       ./7zara10.txt
bash$> ./myfilter -l    ./7zara10.txt
bash$> ./myfilter -w    ./7zara10.txt
bash$> ./myfilter -l -w ./7zara10.txt
bash$> ./myfilter -c    ./7zara10.txt
bash$> ./myfilter -c -i ./7zara10.txt
bash$> ./myfilter
bash$> cat ./7zara10.txt | ./myfilter -l
```

Zum Programm soll eine etwa 3–seitige Dokumentation erstellt werden, in der die Arbeitsweise des Programms kurz skizziert wird.

## 2 Bemerkungen und Tips

Die Entwicklungsumgebung ist *MinGW*. Das Testen sollte in einer *bash*–Umgebung stattfinden. Entweder stellt *MinGW* diese zur Verfügung oder man öffnet eine *Gitbash*, in der man arbeiten kann.

Für das Einlesen der Daten ist die Funktionen `getc()` gegenüber der oftmals instabilen (und komplizierteren) `scanf()`-Funktion vorzuziehen. Zum Einlesen von Optionen und Argumenten gibt es unter Unix die Funktion `getopt()` .

Zur besseren Struktur und um ein besseres Testen zu ermöglichen, unterteilen Sie das gesamte Programm in mehrere Funktionalitäten und testen Sie jeweils nur die einzelnen Funktionen für sich:

- Einlesen und Verarbeitung der Parameter (`getopt()` verstehen!)
- Bestimmen der Ein– und Ausgabe–Kanäle (Kanäle und Umleitungen verstehen!)
- Unterfunktion zur Zählung der Zeichen, Worte und Zeilen (C–Programmierung und Zeiger)

Unter Umständen erzeugen Sie verschiedene Programme, in der nur eine bestimmte Funktion ausprogrammiert wird und fügen diese dann später zusammen.

Testen Sie oft und gründlich und berücksichtigen dabei möglichst viele Testfälle. Gehen Sie auf die von Ihnen durchgeführten Tests auch kurz in der Dokumentation ein.

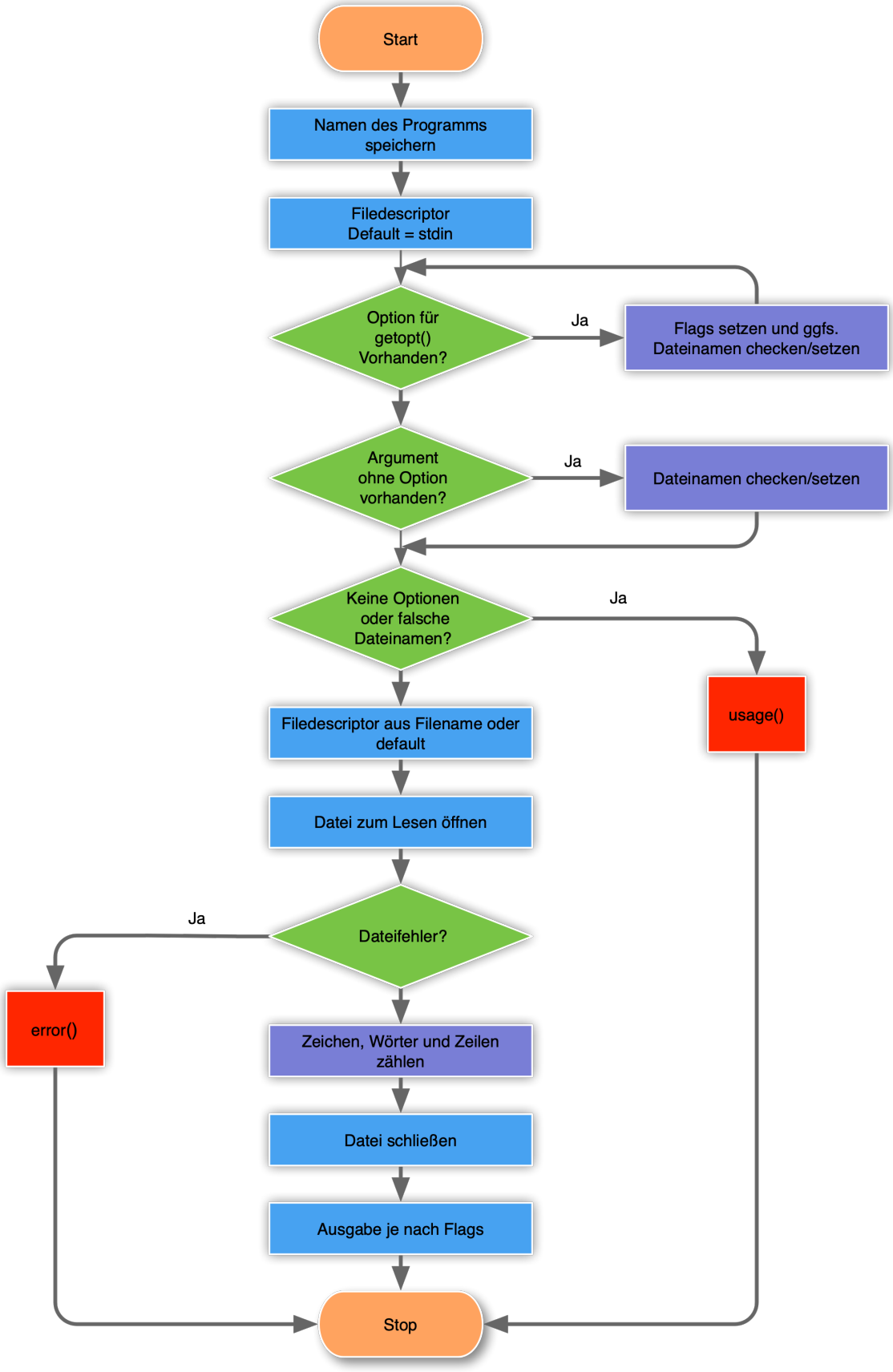
## 3 Abgabe

Die lauffähigen Programme und der Quelltext sind mit einer etwa 3–seitigen Dokumentation abzugeben (ins Repository: C⇒ Eingang⇒ <Name>, Dokumentation in LaTeX oder `ASCII`).

## 4 Zeitrahmen

Der Zeitrahmen dieser Aufgabe ist mit etwa 1–2 Wochen angesetzt.

# 5 Musterlösung



**Musterlösung für** `myfilter.c`

```c
/**
 * myfilter.c
 * Author: Frank Zimmermann
 * Date: 8/2020
 *
 * Example solution for Aufgabe01
 * (see K&R book)
 * (it uses several constructs just for demonstration...)
 * (...with slightly too much comments)
 *
 * Counts chars, words and lines.
 * Program can be used as
 * a filter or with one file as input argument.
 * The output is controlled by option arguments -c,-w,-l
 * The option -i <inputfile> is optional, <inputfile> is sufficient
 * Every C modul should contain such a (or similar) description header!
 */

/**
 * Includes
 */
#include <stdio.h> /* standard io from c */
#include <getopt.h> /* for processing the program arguments */
#include <string.h> /* for string copy */
#include <stdlib.h> /* constants like EXIT_SUCCESS,... */
#include <unistd.h> /* access, ... */

/**
 * Macros
 */
#define FALSE 0   /* just for better readability (alt: <stdbool.h> or FALSE (0!=0)) */
#define TRUE !(FALSE)
#define ERROR_FILEREAD(message) fprintf(stderr,"%s is not a valid input file name!\n",message)
#define ERROR_FILEOPEN(message) fprintf(stderr,"An error occured while opening %s!\n",message)
#define ERROR_USAGE(message) fprintf(stderr,"Usage: %s [-c|-w|-l] [-i] inputfile\n",message)
#define ERROR_NAMELESS(message) fprintf(stderr,"...Nameless and lonly...%s\n",message)

/**
 * Data structure and type definition for a multi value return (for demonstration)
 */
typedef struct {
  long chars; /* counted chars */
  long words; /* counted words */
  long lines; /* counted lines */
} TRIPLET;

/**
 * Forward declarations for C syntax
 */
TRIPLET countCharsWordsLines(FILE *in); /* do the counting work */
void   usage(void);                  /* simple usage message */
FILE* openInputFile(char* str);     /* opens an input file or stdin */
int    checkFilename(char* str);     /* check if filename is readable */

/**
 * Global variables
 * (not necessary, we could use argv[0] directly, just to have a global variable)
 */
```

```
59 char progname[255];                  /* the progname for the usage() function */
60
61
62 /**
63  * every C program needs a main procedure
64  * @param argc argument count
65  * @param argv array of strings
66  * @return     0=Success
67  */
68 int main(int argc, char** argv) {
69   int mainReturn = EXIT_SUCCESS; /* a return for the main function */
70   TRIPLET ret3;                  /* a return for the worker function */
71   int option;                    /* contains one option letter */
72   char filename[255] = "";       /* takes the filname */
73   int filenameValid = TRUE;
74   FILE* in;                      /* the file descriptor */
75
76   enum {OFF, ON};                /* to demonstrate an unamed enum OFF=0, ON=1 */
77
78   struct {                       /* to demonstrate a bit field for the options */
79     unsigned int c: 1;
80     unsigned int w: 1;
81     unsigned int l: 1;
82     unsigned int i: 1;
83   } optflags = {OFF, OFF, OFF, OFF}; /* you HAVE to initialize! */
84
85   in = stdin;                     /* for the case: openInputFile() is never used */
86   strncpy(progname, argv[0], 255); /* progname/path from argument list (unix/win) to global */
87
88   /**
89    * reading the options and arguments and setting flags
90    */
91   while ((option = getopt(argc, argv, "cwli:")) != EOF) {
92     switch (option) {
93     case 'c' :                    /* print counted char */
94       optflags.c = ON;            /* saving in a bit field */
95       break;
96     case 'w' :                    /* print counted words */
97       optflags.w = ON;            /* saving in a bit field */
98       break;
99     case 'l' :                    /* print counted lines */
100      optflags.l = ON;            /* saving in a bit field */
101      break;
102    case 'i' :                    /* get input filename from argument list */
103      optflags.i = ON;            /* saving in a bit field */
104      strncpy(filename, optarg, 255);
105      filenameValid = checkFilename(filename);/* just check file */
106      break;
107    default :
108      usage();                    /* in case of wrong call of program */
109      exit(EXIT_FAILURE);
110    }
111  }
112
113  /**
114   * looking for a further parameter without option
115   */
116  if (optind < argc) {                   /* more arguments than options */
117    strncpy(filename, argv[optind], 255);
118    filenameValid = checkFilename(filename); /* check file. */
119  }
```

```
120
121   /**
122    * test for no options at all or wrong filename
123    */
124   if (filenameValid == FALSE || (optflags.c == OFF && optflags.w == OFF && optflags.l == OFF)) {
125     usage();
126     exit(EXIT_FAILURE);                  /* no arguments or wrong filename */
127   }
128
129   /**
130    * when we are here: filenameValid is TRUE AND there are valid flags
131    * openInputFile gives a new pointer to FILE (when filename has content)
132    * or returns stdin (when filename is empty)
133    * openInputFile returns NULL in case of error
134    */
135   mainReturn = ((in = openInputFile(filename)) == NULL) ? EXIT_FAILURE : EXIT_SUCCESS;
136
137   if (mainReturn == EXIT_SUCCESS) {      /* if no errors in opening */
138     ret3 = countCharsWordsLines(in);     /* this does all the work */
139     fclose(in);                          /* close file, for stdin not harmful */
140
141     /**
142      * see what we have to write to stdout:
143      */
144     if (optflags.c == ON) {
145       printf("Zeichen␣=␣%ld\n", ret3.chars);
146     }
147     if (optflags.w == ON) {
148       printf("Wörter␣=␣%ld\n", ret3.words);
149     }
150     if (optflags.l == ON) {
151       printf("Zeilen␣=␣%ld\n", ret3.lines);
152     }
153     fflush(stdout); /* better be sure */
154   }
155   else {
156     ERROR_FILEOPEN(filename);
157   }
158   return mainReturn;
159 }
160
161 /**
162  * counts chars,words and lines
163  * @param in pointer to FILE
164  * @return   TRIPLET
165  */
166 TRIPLET countCharsWordsLines(FILE *in) {
167   TRIPLET retval = {0, 0, 0};
168   long nl,               /* counter for lines */
169        nw,               /* counter for words */
170        nc;               /* counter for chars */
171   int  state,            /* flag for position */
172        c;                /* a char */
173   /* needful definitions (for demonstration) */
174   enum {OUT, IN};
175
176   state = OUT;           /* initialize position flag */
177   nc = nw = nl = 0;      /* initialize values */
178   while ((c = getc(in)) && !feof(in)) {
179     ++nc;                     /* increment character count */
180     if (c == '\n') {
```

```
181       ++nl;              /* increment line count */
182     }
183     if (c == ' ' || c == '\n' || c == '\t') {
184       state = OUT;       /* leaving a word */
185     }
186     else if (state == OUT) {
187       state = IN;        /* entering a word */
188       ++nw;              /* ...and count */
189     }
190   }
191   fclose(in);
192 
193   retval.chars = nc;     /* fill struct fields ... */
194   retval.words = nw;
195   retval.lines = nl;
196   return retval;         /* return the struct */
197 }
198 
199 /**
200  * just print out a usage() message on stderr
201  */
202 void usage() {
203   if (*progname != '\0') {
204     ERROR_USAGE(progname);
205   }
206   else {
207     ERROR_NAMELESS("Greeting to grandma...!"); /* what is this ??! */
208   }
209 }
210 
211 /**
212  * Tries to open a file with a string
213  * @param str filename
214  * @return    pointer to FILE or NULL
215  */
216 FILE* openInputFile(char* str) {
217   FILE* in = stdin;
218   if (*str != '\0') {           /* str is empty */
219     if ((in = fopen(str, "r")) == NULL) {
220       ERROR_FILEREAD(str);      /* could not open for read */
221     }
222   }
223   return in;
224 }
225 
226 /**
227  * just checks the readability and print on stderr in case of error
228  * @param str string holds filename
229  * @return    boolean value for success
230  */
231 int checkFilename(char* str) {
232   int retVal;
233   if ( access( str, R_OK ) == EXIT_SUCCESS) { /* security issue...! */
234       retVal = TRUE;
235   }
236   else {
237     retVal = FALSE;
238     ERROR_FILEREAD(str);
239   }
240   return retVal;
241 }
```

**Musterlösung für** `Makefile`

```
CC = gcc
OBJ = myfilter.o
CFLAGS = -Wall
TEXFILE = aufgabe01
LATEX = latexmk
LATEXFLAGS = -pdf -silent -interaction=nonstopmode
LATEXTMP = *.log *.listing *.fls *.fdb_latexmk *.synctex.gz *.aux

all: aufgabe01.pdf myfilter

aufgabe01.pdf: aufgabe01.tex myfilter.c Aufgabe01PAP.png Makefile
	$(LATEX) $(LATEXFLAGS) $(TEXFILE).tex && open $(TEXFILE).pdf

myfilter: $(OBJ)
	$(CC) $(CFLAGS) -o myfilter myfilter.o

myfilter.o: myfilter.c
	$(CC) $(CFLAGS) -c myfilter.c

clean:
	rm -f $(OBJ) $(LATEXTMP)
```